**Exploring handwritten digit classification: a tidy analysis of the MNIST dataset**

January 22, 2018

By [David Robinson](https://www.r-bloggers.com/author/david-robinson/)

[This article was first published on [**Variance Explained**](http://varianceexplained.org/r/digit-eda/), and kindly contributed to [R-bloggers](https://www.r-bloggers.com/)]. (You can report issue about the content on this page [here](https://www.r-bloggers.com/contact-us/))

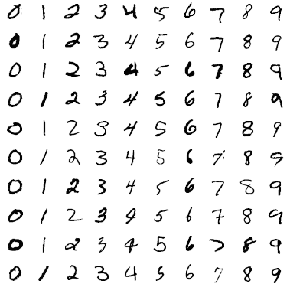
Want to share your content on R-bloggers? [click here](https://www.r-bloggers.com/add-your-blog/) if you have a blog, or [here](https://r-posts.com/) if you don't.

[Share](https://www.facebook.com/sharer.php?u=https%3A%2F%2Fwww.r-bloggers.com%2Fexploring-handwritten-digit-classification-a-tidy-analysis-of-the-mnist-dataset%2F)[Tweet](https://twitter.com/intent/tweet?text=Exploring%20handwritten%20digit%20classification%3A%20a%20tidy%20analysis%20of%20the%20MNIST%20dataset&url=https://www.r-bloggers.com/exploring-handwritten-digit-classification-a-tidy-analysis-of-the-mnist-dataset/&via=Rbloggers)

In a [recent post](http://varianceexplained.org/r/ds-ml-ai/), I offered a definition of the distinction between data science and machine learning: that data science is focused on extracting insights, while machine learning is interested in making predictions. I also noted that the two fields greatly overlap:

I use both machine learning and data science in my work: I might fit a model on Stack Overflow traffic data to determine which users are likely to be looking for a job (machine learning), but then construct summaries and visualizations that examine why the model works (data science). This is an important way to discover flaws in your model, and to combat algorithmic bias. This is one reason that data scientists are often responsible for developing machine learning components of a product.

I’d like to further explore how data science and machine learning complement each other, by demonstrating how I would use data science to approach a problem of image classification. We’ll work with a classic machine learning challenge: the [MNIST digit database](https://en.wikipedia.org/wiki/MNIST_database).



The challenge is to classify a handwritten digit based on a 28-by-28 black and white image. MNIST is often credited as one of the first datasets to prove the effectiveness of neural networks.

In a series of posts, I’ll be training classifiers to recognize digits from images, while using data exploration and visualization to build our intuitions about why each method works or doesn’t. Like most of my posts I’ll be analyzing the data through tidy principles, particularly using the dplyr, tidyr and ggplot2 packages. In this first post we’ll focus on **exploratory data analysis**, to show how you can better understand your data before you start training classification algorithms or measuring accuracy. This will help when we’re choosing a model or transforming our features.

**Preprocessing**

The default MNIST dataset is somewhat inconveniently formatted, but [Joseph Redmon has helpfully created a CSV-formatted version](https://pjreddie.com/projects/mnist-in-csv/). We can download it with the readr package.

library(readr)  
library(dplyr)  
  
mnist\_raw <- read\_csv("<https://pjreddie.com/media/files/mnist_train.csv>", col\_names = FALSE)

mnist\_raw[1:10, 1:10]

## # A tibble: 10 x 10  
## X1 X2 X3 X4 X5 X6 X7 X8 X9 X10  
##   
## 1 5 0 0 0 0 0 0 0 0 0  
## 2 0 0 0 0 0 0 0 0 0 0  
## 3 4 0 0 0 0 0 0 0 0 0  
## 4 1 0 0 0 0 0 0 0 0 0  
## 5 9 0 0 0 0 0 0 0 0 0  
## 6 2 0 0 0 0 0 0 0 0 0  
## 7 1 0 0 0 0 0 0 0 0 0  
## 8 3 0 0 0 0 0 0 0 0 0  
## 9 1 0 0 0 0 0 0 0 0 0  
## 10 4 0 0 0 0 0 0 0 0 0

This dataset contains one row for each of the 60000 training instances, and one column for each of the 784 pixels in a 28 x 28 image. The data as downloaded doesn’t have column labels, but are arranged as “row 1 column 1, row 1 column 2, row 1 column 3…” and so on). This is a useful enough representation for machine learning. But as Jenny Bryan often discusses, we shouldn’t feel constricted by our current representation of the data, and for exploratory analysis we may want to make a few changes.

* **We’d like to represent it as one-row-per-pixel-per-instance**. It would be challenging to visualize this default data as images, but it becomes easier once we consider each pixel an “observation”.
* **We’d like custom features that have meaning within the problem** For instance, we don’t just have 784 arbitrary features; we have 28 rows and 28 columns. So rather than having labels like “X2” and “X3”, we’d like to keep track of variables for **x** and **y** (as coordinate positions of the images).
* **We’d like to explore a subset first**. While you’re first exploring data, you don’t need your full complement of training examples, since working with a subset lets you iterate quickly and create proof of concepts while saving computational time.

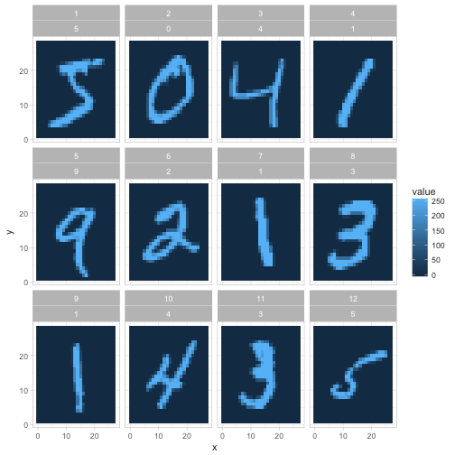
With that in mind, we’ll [gather](http://tidyr.tidyverse.org/) the data, do some arithmetic to keep track of x and y within an image, and keep only the first 10,000 training instances.

library(tidyr)  
  
pixels\_gathered <- mnist\_raw %>%  
 head(10000) %>%  
 rename(label = X1) %>%  
 mutate(instance = row\_number()) %>%  
 gather(pixel, value, -label, -instance) %>%  
 tidyr::extract(pixel, "pixel", "(\\d+)", convert = TRUE) %>%  
 mutate(pixel = pixel - 2,  
 x = pixel %% 28,  
 y = 28 - pixel %/% 28)  
  
pixels\_gathered

## # A tibble: 7,840,000 x 6  
## label instance value pixel x y  
##   
## 1 5 1 0 0 0 28.0  
## 2 0 2 0 0 0 28.0  
## 3 4 3 0 0 0 28.0  
## 4 1 4 0 0 0 28.0  
## 5 9 5 0 0 0 28.0  
## 6 2 6 0 0 0 28.0  
## 7 1 7 0 0 0 28.0  
## 8 3 8 0 0 0 28.0  
## 9 1 9 0 0 0 28.0  
## 10 4 10 0 0 0 28.0  
## # ... with 7,839,990 more rows

We now have one row for each pixel in each image. This is a useful format because it lets us visualize the data along the way. For example, we can visualize the first 12 instances with a couple lines of ggplot2.

library(ggplot2)  
theme\_set(theme\_light())  
  
pixels\_gathered %>%  
 filter(instance <= 12) %>%  
 ggplot(aes(x, y, fill = value)) +  
 geom\_tile() +  
 facet\_wrap(~ instance + label)



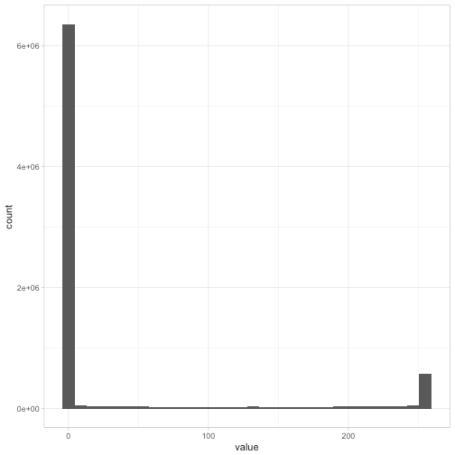
We’ll still often return to the one-row-per-instance format (especially once we start training classifiers in future posts), but this is a fast way to understand and appreciate how the data and the problem is structured. In the rest of this post we’ll also polish this kind of graph (like making it black and white rather than a scale of blues).

**Exploring pixel data**

Let’s get more comfortable with the data. From the legend above, it looks like 0 represents blank space (like the edges of the image), and a maximum around 255 represents the darkest points of the image. Values in between may represent different shades of “gray”.

How much gray is there in the set of images?

ggplot(pixels\_gathered, aes(value)) +  
 geom\_histogram()



Most pixels in the dataset are completely white, along with another set of pixels that are completely dark, with relatively few in between. If we were working with black-and-white *photographs* (like of faces or landscapes), we might have seen a lot more variety. This gives us a hint for later feature engineering steps: if we wanted to, we could probably replace each pixel with a binary 0 or 1 with very little loss of information.

I’m interested in how much variability there is within each digit label. Do all 3s look like each other, and what is the “most typical” example of a 6? To answer this, we can find the *mean* value for each position within each label, using dplyr’s group\_by and summarize.

pixel\_summary <- pixels\_gathered %>%  
 group\_by(x, y, label) %>%  
 summarize(mean\_value = mean(value)) %>%  
 ungroup()  
  
pixel\_summary

We visualize these average digits as ten separate facets.

library(ggplot2)  
  
pixel\_summary %>%  
 ggplot(aes(x, y, fill = mean\_value)) +  
 geom\_tile() +  
 scale\_fill\_gradient2(low = "white", high = "black", mid = "gray", midpoint = 127.5) +  
 facet\_wrap(~ label, nrow = 2) +  
 labs(title = "Average value of each pixel in 10 MNIST digits",  
 fill = "Average value") +  
 theme\_void()